浙江大学

本科实验报告

课程名称：计算机组成

姓名：曾一欣
学院：竺可桢学院
班级：求是科学班（计算机科学与技术）
专业：计算机科学与技术
学号：3180105144
指导老师：姜晓红

2020.3.15

浙江大学实验报告

课程名称： 计算机组成　　实验类型：综合

实验项目名称：Lab1 part1- Multiplex & Lab2 – Display module

学生姓名：曾一欣　　专业：计算机科学与技术　　学号： 3180105144

同组学生姓名： None　　指导老师： 姜晓红

实验地点：无　　实验日期： 2020 年 3 月 15 日

一、 实验目的和要求

Lab1

Objective

1. Learn the tools and procedure of EDA development
2. Warm up:　review logical design methods
3. Implement and package basic function unit
4. Set a validation environment for the lab
5. Implement some basic modules of computer system.

Requirement

1. Review labs in the course of Digital Logic Design - Multiplexer
2. Review the module of output - debouncing circuite
3. Implement memory IP core - 32 bit ROM、32 bit RAM
4. Implement display channel

Lab2

Objective

1. Understand IO device and interface
2. Understand the simplest interface GPIO
3. Understand how to implement simple IO

Requirement

1. Optimize the display module in digital logic design - 7-segment tube display in graph mode
2. Modify the display channel mode - Channel 1 will be set to CPU
   The other channels will be used for test

二、 实验内容和原理

1/8 Multiplex : Multi_8CH32

function

Chose one from 8 channel for 7-segment digital tube display.

input

• clk : synchronous clock

• rst : reset signal

• EN : enable signal for channel 0

• SW(7 : 5) : channel selection control signal

• Point_in(63 : 0) : point of 7-segment tube

  –8 bit for one channel, so 8 bit * 8 channel = 64 bit

• LES(63 : 0) : blink enable signal

  –8 bit for one channel, so 8 bit * 8 channel = 64 bit

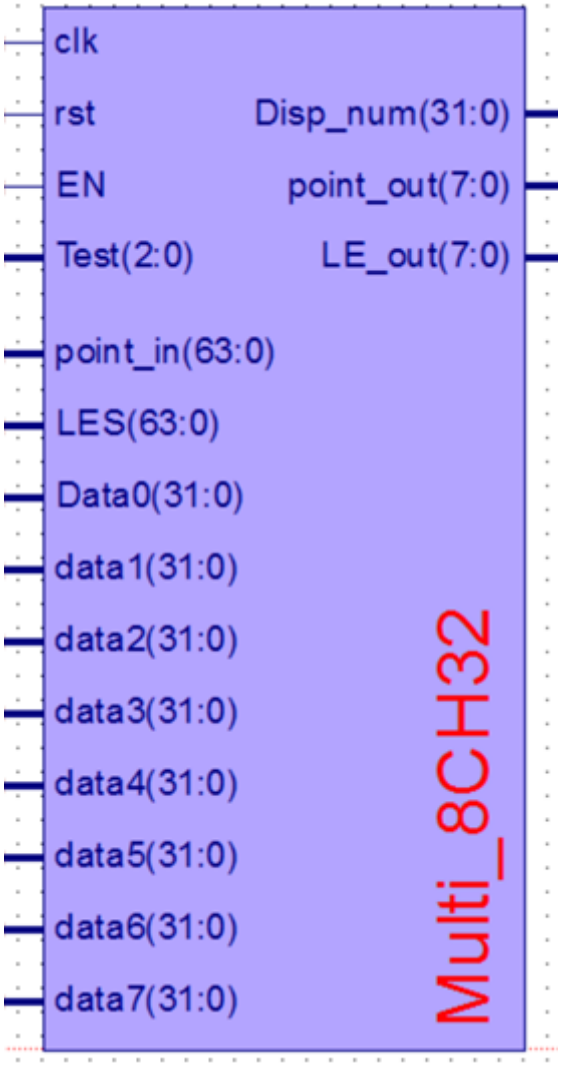
•Data 0 - Data 7[31 : 0] : data input channel

output

• Disp_num (31 : 0) : the chosen data output

• LES_out(7 : 0) : enable output

• Point_out(7 : 0) : point output



And the main code is shown in the experiment procedure part.

SSeg7_Dev

function

Transform the hexs signal calculated by the other module into output signal that can be recognize by 7-segment device.

input

• 32 bit binary data in hexs

• 1 bit SW0: When SW[0]=1， display 8 Hexs digital num: high 16 bits when SW[1]=1, low 16 bits when SW[1]=0. When SW[0]=0， display 7-segment LED point array.

• Blink frequency is provide by U8(Div[25])

• Start: signal for beginning scan

• point: 7-segment point

• LES: blink enable

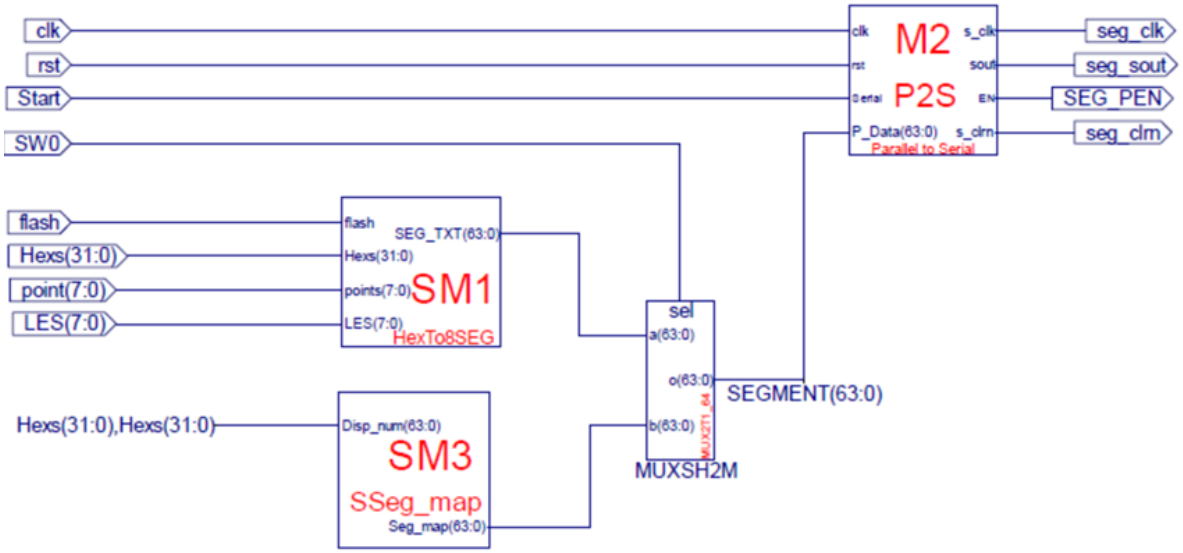
output

• seg_clk: clock

• seg_out: display number

• SEG_PEN: enable

• seg_clrn: clear to zero



And the main code is shown in the experiment procedure part.

SPIO

function

produce control signal for the LED display part according to the input message.

input

• P_Data: the information that controls the LED devices

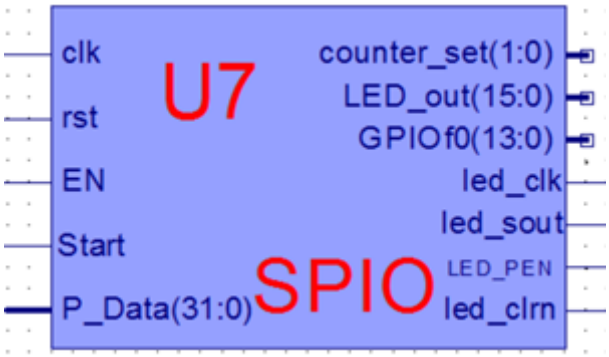
• clk: clock signal, EN: enable signal, Start: start signal, rst: reset signal

output

• serial output: led_clk - clock signal, led_sout - serial output data,

LED_PEN - enable signal, led_clrn - reset signal

• parallel output: LED_out, counter_set, GPIOf0



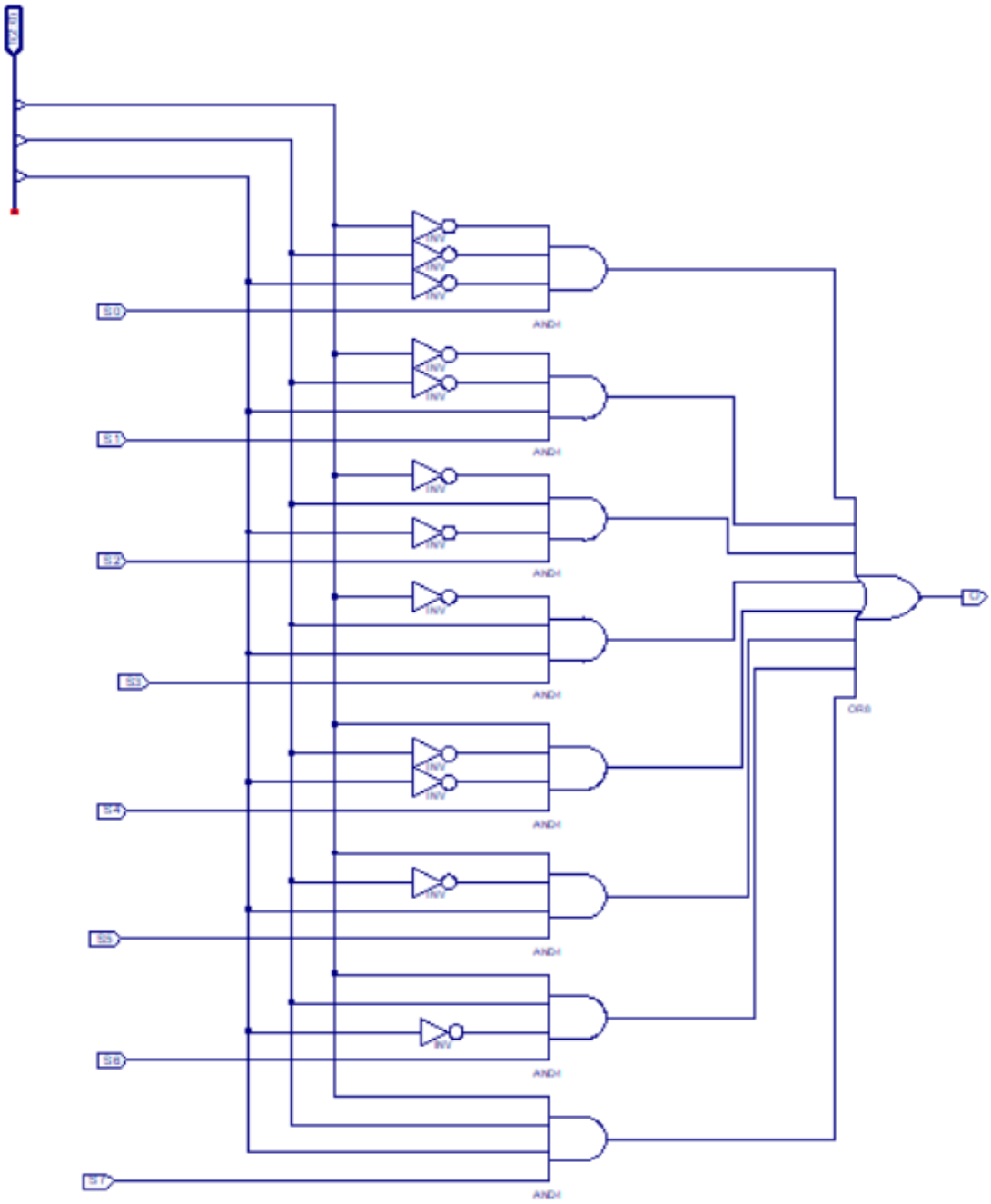
And the main code is shown in the next part.

三、 实验过程和数据记录及结果分析

1/8 Multiplex : Multi_8CH32

MUX8T1_1

designed by sch:



MUX8T1_8

designed with verilog language:

```
module MUX8T1_8(
    input [2:0]s,
    input [7:0]I0,
    input [7:0]I1,
    input [7:0]I2,
    input [7:0]I3,
    input [7:0]I4,
    input [7:0]I5,
    input [7:0]I6,
    input [7:0]I7,
    output [7:0]o
    );

MUX8T1_1 m0(.T(s),
            .S0(I0[0]),
```

```
            .S1(I1[0]),
            .S2(I2[0]),
            .S3(I3[0]),
            .S4(I4[0]),
            .S5(I5[0]),
            .S6(I6[0]),
            .S7(I7[0]),
            .O(o[0]));

... //omit

MUX8T1_1 m7(.T(s),
            .S0(I0[7]),
            .S1(I1[7]),
            .S2(I2[7]),
            .S3(I3[7]),
            .S4(I4[7]),
            .S5(I5[7]),
            .S6(I6[7]),
            .S7(I7[7]),
            .O(o[7]));

endmodule
```

MUX8T1_32

designed with verilog language:

```
module MUX8T1_32(
    input [2:0]s,
    input [31:0]I0,
    input [31:0]I1,
    input [31:0]I2,
    input [31:0]I3,
    input [31:0]I4,
    input [31:0]I5,
    input [31:0]I6,
    input [31:0]I7,
    output [31:0]o
    );

MUX8T1_8
m1(.s(s),.I0(I0[7:0]),.I1(I1[7:0]),.I2(I2[7:0]),.I3(I3[7:0]),.I4(I4[7:0]),
            .I5(I5[7:0]),.I6(I6[7:0]),.I7(I7[7:0]));

... //omit

MUX8T1_8 m4(.s(s),.I0(I0[31:24]),.I1(I1[31:24]),.I2(I2[31:24]),.I3(I3[31:24]),
            .I4(I4[31:24]),.I5(I5[31:24]),.I6(I6[31:24]),.I7(I7[31:24]));

endmodule
```

Multi_8CH32

designed with verilog language:

```
module Multi_8CH32(
    input clk,
    input rst,
    input EN,
    input [2:0]Test,
    input [63:0]point_in,
    input [63:0]LES,
    input [31:0]Data0,
    input [31:0]Test_Data1,
    input [31:0]Test_Data2,
    input [31:0]Test_Data3,
    input [31:0]Test_Data4,
    input [31:0]Test_Data5,
    input [31:0]Test_Data6,
    input [31:0]Test_Data7,
    output [7:0]point_out,
    output [7:0]LE_out,
    output [31:0]Disp_num
    );

reg[31:0] disp_data=32'hAA5555AA;
reg[7:0] cpu_blink=8'b11111111,cpu_point=4'b00000000;

MUX8T1_32 MUX1_DispData(.I0(disp_data),.I1(Test_Data1),.I2(Test_Data2),
                        .I3(Test_Data3),.I4(Test_Data4),.I5(Test_Data5),
                        .I6(Test_Data6),.I7(Test_Data7),.o(Disp_num));

MUX8T1_8 MUX2_Blink(.I0(cpu_blink),.I1(LES[15:8]),.I2(LES[23:16]),
                    .I3(LES[31:24]),.I4(LES[39:32]),.I5(LES[47:40]),
                    .I6(LES[55:48]),.I7(LES[63:56]),.o(LE_out));

MUX8T1_8 MUX2_Point(.I0(cpu_point),.I1(point_in[15:8]),.I2(point_in[23:16]),

.I3(point_in[31:24]),.I4(point_in[39:32]),.I5(point_in[47:40]),
                    .I6(point_in[55:48]),.I7(point_in[63:56]),.o(point_out));

always@(posedge clk)begin
    if(EN)begin
        disp_data <= Data0;
        cpu_blink <= LES[7:0];
        cpu_point <= point_in[7:0];
    end
    else begin
        disp_data <= disp_data;
        cpu_blink <= cpu_blink;
        cpu_point <= cpu_point;
    end
end

endmodule
```

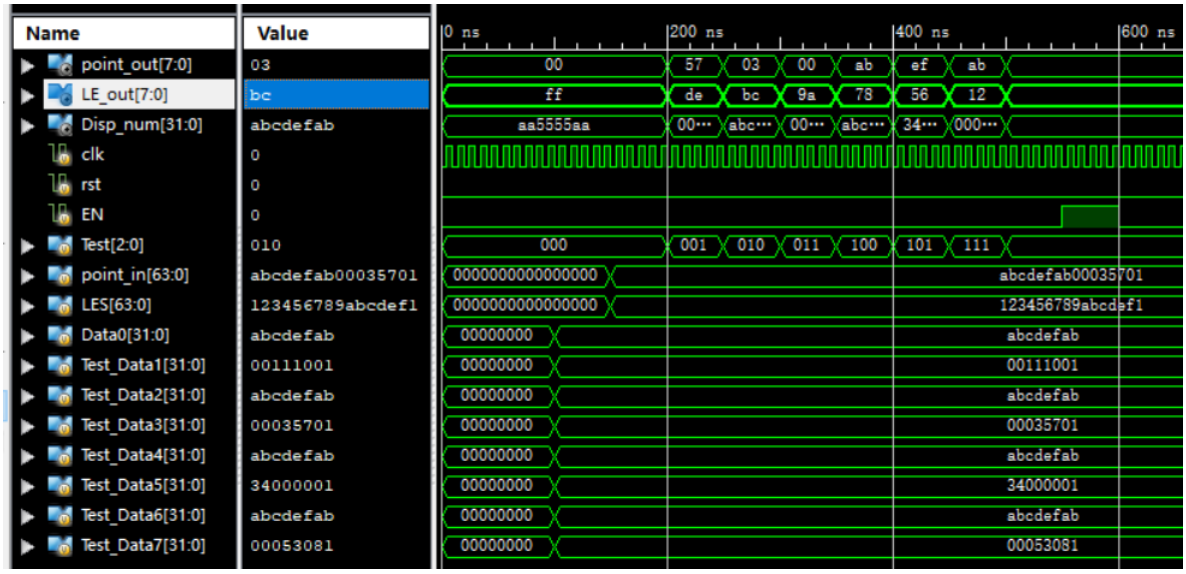
Simulation

test code:

```
    initial begin
        clk = 0;
```

```
        rst = 0;
        EN = 0;
        Test = 0;
        point_in = 0;
        LES = 0;
        Data0 = 0;
        Test_Data1 = 0;
        Test_Data2 = 0;
        Test_Data3 = 0;
        Test_Data4 = 0;
        Test_Data5 = 0;
        Test_Data6 = 0;
        Test_Data7 = 0;

        // Wait 100 ns for global reset to finish
        #100;

        Data0 = 32'hABCDEFAB;
        Test_Data1 = 32'h00111001;
        Test_Data2 = 32'hABCDEFAB;
        Test_Data3 = 32'h00035701;
        Test_Data4 = 32'hABCDEFAB;
        Test_Data5 = 32'h34000001;
        Test_Data6 = 32'hABCDEFAB;
        Test_Data7 = 32'h00053081;
        #50;
        point_in = 64'hABCDEFAB00035701;
        LES =      64'h123456789ABCDEF1;
        #50;
        Test = 3'b001;#50;
        Test = 3'b010;#50;
        Test = 3'b011;#50;
        Test = 3'b100;#50;
        Test = 3'b101;#50;
        Test = 3'b111;#50;
        Test = 3'b010;#50;
        EN = 1;#50;
        EN = 0;#50;

    end

    always begin
    clk = 0; #5;
    clk = 1; #5;
    end

endmodule
```

test result:



From the shown value above we can confirm that Multi_8CH32 do its job correctly.

SSeg7_Dev

MC14495_ZJU

Provided by the .sym file of my_MC14495 completed in the digital logical experiment.

HexTo8SEG

```
module HexTo8SEG(
    input flash,
    input [7:0]LES,
    input [7:0]point,
    input [31:0]Hexs,
    output [63:0]SEG_TXT
    );

MC14495_ZJU m0(.D0(Hexs[0]),
               .D1(Hexs[1]),
               .D2(Hexs[2]),
               .D3(Hexs[3]),
               .LE((flash&LES[0])),
               .point(point[0]),
               .a(SEG_TXT[0]),
               .b(SEG_TXT[1]),
               .c(SEG_TXT[2]),
               .d(SEG_TXT[3]),
               .e(SEG_TXT[4]),
               .f(SEG_TXT[5]),
               .g(SEG_TXT[6]),
               .p(SEG_TXT[7]));

...//omit

MC14495_ZJU m7(.D0(Hexs[28]),
               .D1(Hexs[29]),
               .D2(Hexs[30]),
               .D3(Hexs[31]),
               .LE((flash&LES[7])),
               .point(point[7]),
               .a(SEG_TXT[56]),
```

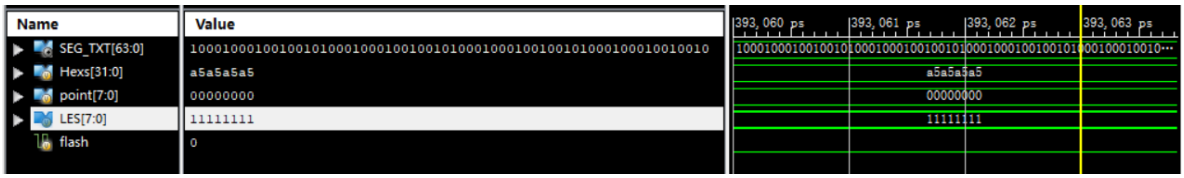
```
                .b(SEG_TXT[57]),
                .c(SEG_TXT[58]),
                .d(SEG_TXT[59]),
                .e(SEG_TXT[60]),
                .f(SEG_TXT[61]),
                .g(SEG_TXT[62]),
                .p(SEG_TXT[63]));

endmodule
```

Simulation for HexTo8SEG

test code:

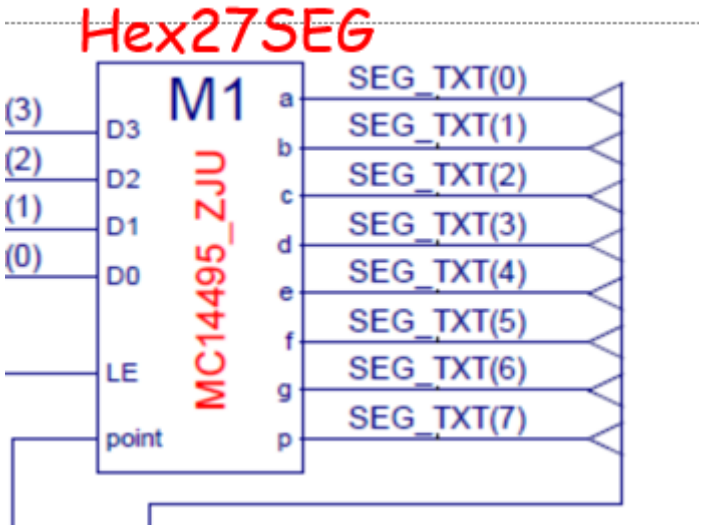
```
    initial begin
        // Initialize Inputs
        Hexs = 0;
        point = 0;
        LES = 8'b11111111;
        flash = 1;

        #100;
        flash = 0;

        Hexs = 32'h12345678;
        #100;

        Hexs = 32'hA5A5A5A5;

    end

endmodule
```

test result:



From the picture we can confirm that this module works well.

Here comes my first question: signal a-p should connect to SEG[0-7] or SEG[7-0]? From my previous work and the example of HEX_27SEG on the slides (the picture is shown below), it's apparently a is wired to SEG[0]. However from the simulation result we can find that value SEG_TXT is the converse of correct output every 8 bits. So it seems that a should be wired to p.



I think just figure out the rule of high bit in verilog language and the rule of simulation display is not enough to decide which connection method is right. I have to get this module onto the circuit board to find the answer.

Seg_Map

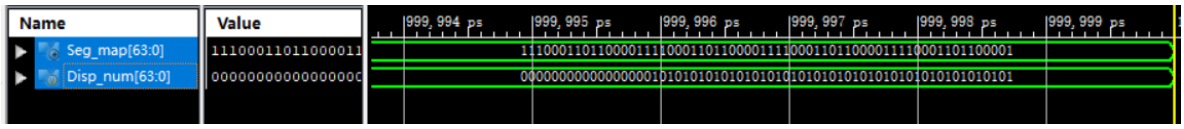
```
module Seg_Map(
    input [63:0]Disp_num,
    output [63:0]Seg_map
    );

assign Seg_map={Disp_num[0],Disp_num[4],Disp_num[16],Disp_num[25],
                Disp_num[17],Disp_num[5],Disp_num[12],Disp_num[24],
                Disp_num[1],Disp_num[6],Disp_num[18],Disp_num[27],
                Disp_num[19],Disp_num[7],Disp_num[13],Disp_num[26],
                Disp_num[2],Disp_num[8],Disp_num[20],Disp_num[29],
                Disp_num[21],Disp_num[9],Disp_num[14],Disp_num[28],
                Disp_num[3],Disp_num[10],Disp_num[22],Disp_num[31],
                Disp_num[23],Disp_num[11],Disp_num[15],Disp_num[30],


                Disp_num[0],Disp_num[4],Disp_num[16],Disp_num[25],
                Disp_num[17],Disp_num[5],Disp_num[12],Disp_num[24],
                Disp_num[1],Disp_num[6],Disp_num[18],Disp_num[27],
                Disp_num[19],Disp_num[7],Disp_num[13],Disp_num[26],
                Disp_num[2],Disp_num[8],Disp_num[20],Disp_num[29],
                Disp_num[21],Disp_num[9],Disp_num[14],Disp_num[28],
                Disp_num[3],Disp_num[10],Disp_num[22],Disp_num[31],
                Disp_num[23],Disp_num[11],Disp_num[15],Disp_num[30]};

endmodule
```

Simulation for Seg_Map

test code:

```
    initial begin
        // Initialize Inputs
        Disp_num = 64'b0101010101010101010101010101010101010101010101010101010101010101;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end

endmodule
```

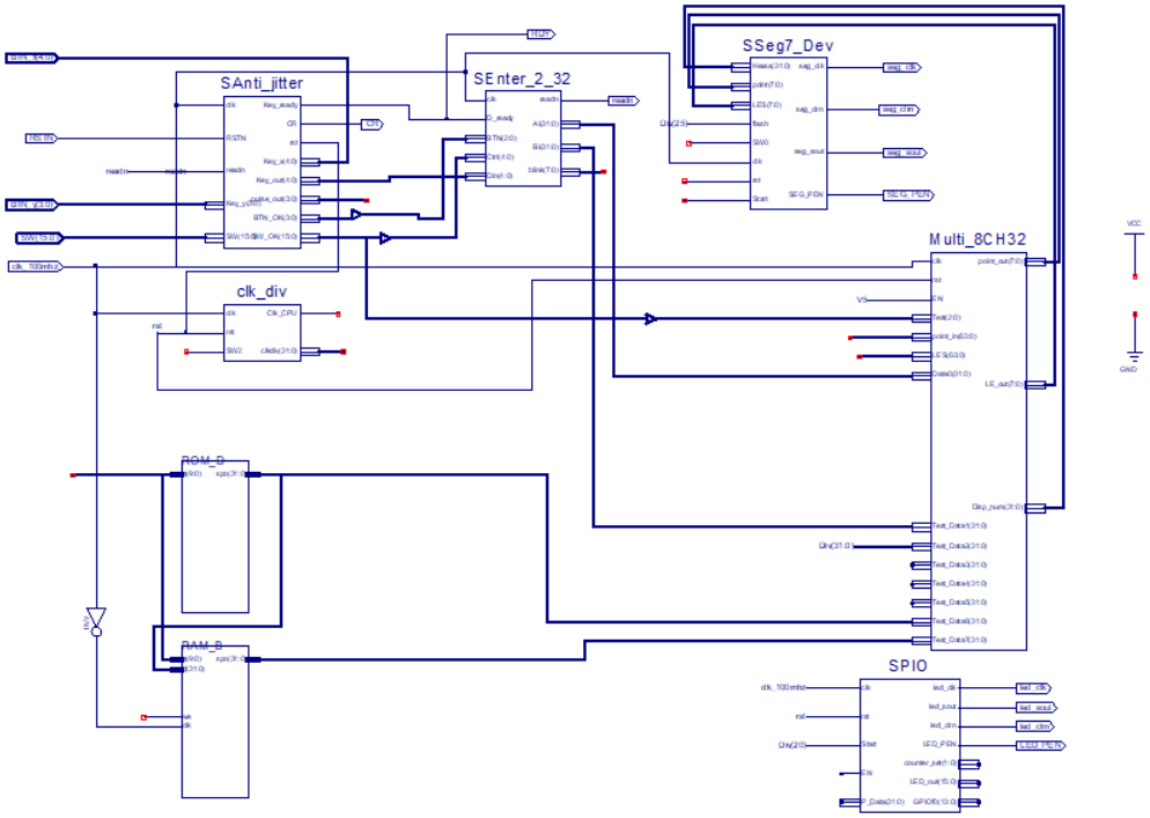
test result:



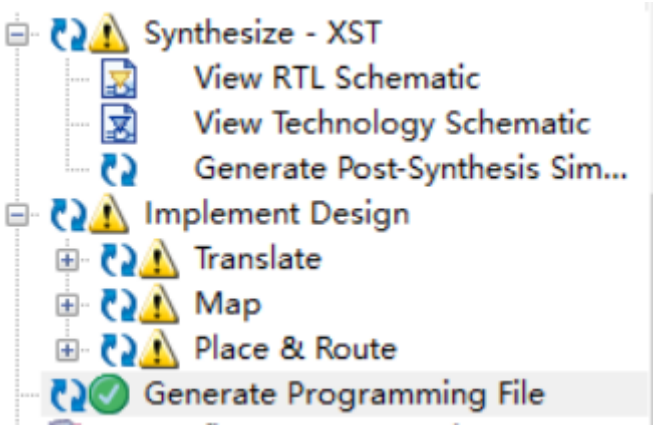
SPIO

```
module SPIO(
    input clk,rst,Start,EN,
    input [31:0]P_Data,
    output reg[1:0]counter_set,
    output [15:0]LED_out,
    output wire led_clk,
    output wire led_sout,
    output wire led_clrn,
    output wire LED_PEN,
    output reg[13:0]GPIOf0
    );

reg [15:0]LED;
assign LED_out=LED;
always@(negedge clk or posedge rst)begin
    if(rst)begin LED <= 8'h2A; counter_set <= 2'b00; end
    else if(EN){GPIOf0[13:0],LED,counter_set} <= P_Data;
    else begin LED<=LED; counter_set <= counter_set; end
end

LED_P2S #(.DATA_BITS(16),.DATA_COUNT_BITS(4))
            LED_PS2(clk,rst,Start,
                {~{LED[0],LED[1],LED[2],LED[3],LED[4],LED[5],LED[6],LED[7],
                   LED[8],LED[9],LED[10],LED[11],LED[12],
                   LED[13],LED[14],LED[15]}},
                led_clk,led_clrn,led_sout,LED_PEN);

endmodule
```

top module



As I can't do simulation on it, I just try to generate it to roughly check it correctness.



The warning is mostly about my version of ise, so I thought the project is possibly without bugs.

四、 讨论与心得

By complete this experiment, I successfully get familiar into verilog language and xilinx ise again. During the procedure of the experiment, I also met some difficulties and confusions. One of them is explained before and I left it there because I can't solve it now. Beside that, I find myself forget how to adjust the ise windows and how to renew the symbols if I changed the original code of the .sym file. And I pay a lot of time on drawing schematic file as well as I am unsure of some details of drawing.

In the word, the aim of this lab task is reached. Hopefully the project could perform well on the board when we get back to school and I could use it for the following experiments.